# ThinkPHP框架\_02

# 今日目标13

* 能够理解参数绑定和依赖注入

public function fun1(Request $request,int $id)

* 能够理解Request请求类的基本使用

$request 依赖注入过来的request类

门面 Request

函数 input 字符串 ?

get put delete post param has only except

* 能够返回json格式数据

return json($data,http状态码,[头信息]);

* 能够进行模板的渲染

return view();

如果是patinfo可以不写，默认就是方法名就是模板名称

自定义路由

return view(模块@控制器/方法)

赋值

return view(‘’,compact(‘变量名不要$’));

* 能够在模板中进行变量输出

默认 标记符号为 {} 变量 {$变量}

可以用函数 {:函数名(变量)}

* 能够在模板中使用foreach等内置标签

{foreach $data as $val}{/foreach}

{volist id=”val” name=”data”}{/volist}

* 能够配置数据库信息

config/database.php

* 能够定义TP5框架中的模型

php think make:model 【模块名】/模型名 模型名为表名相当

* 能够熟练使用数据查询方法

find

select

findOrFail

selectOrFail

where

* 能够完成数据的添加操作

create静态方法

* 能够完成烽据的修改操作

update 静态方法

Articles::where()->update()

* 能够完成数据的删除操作

delete

destroy 静态方法

* 能够理解trait并实现软删除功能

trait 解决多继承 php5.4

数据表中添加一个标识位

# 一、请求

当浏览器向Web服务器发出请求时，它向服务器传递了一个数据块，也就是请求信息。在Thinkphp5.1中，获取请求对象数据，是由**think\Request**类负责，在很多场合下并不需要实例化调用，通常使用依赖注入即可，在其它场合（例如模板输出等）则可以使用think\facade\Request静态类操作。

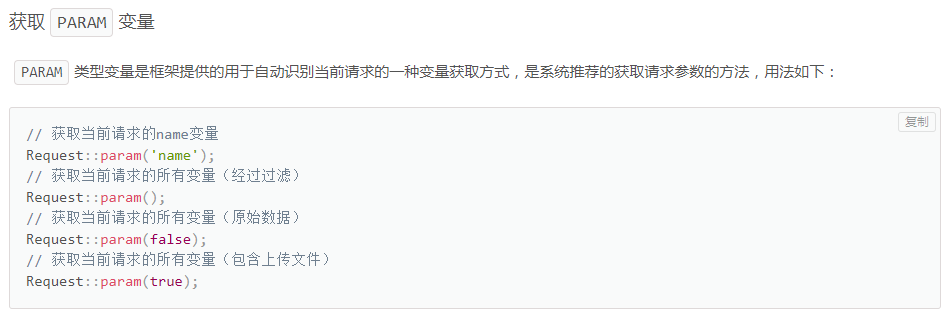
Request::param('name');

$request->param('name');

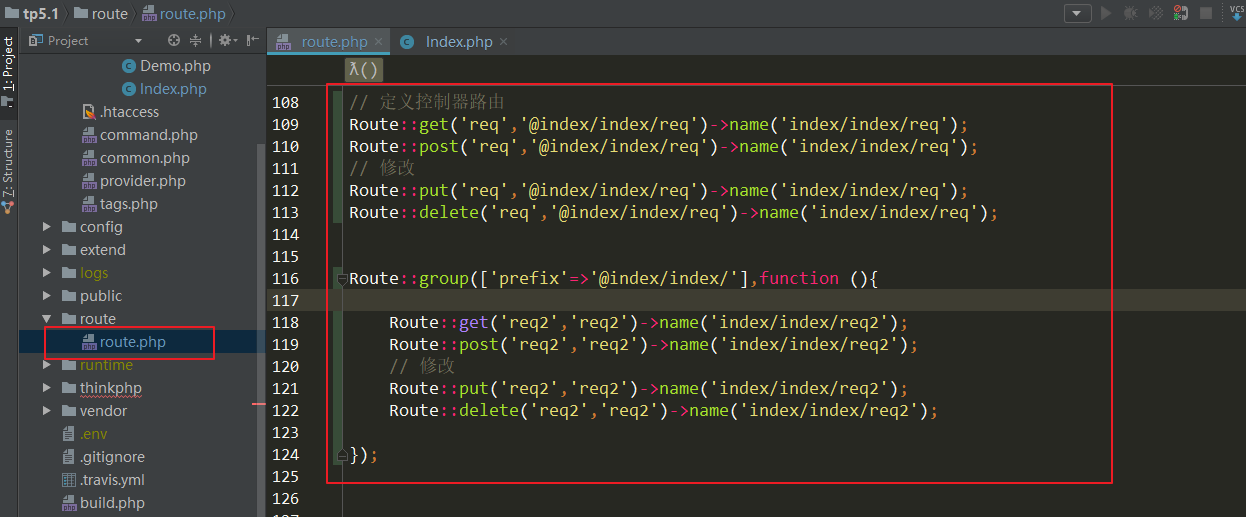
## 1.1、变量获取

变量类型方法('变量名/变量修饰符','默认值','过滤方法')



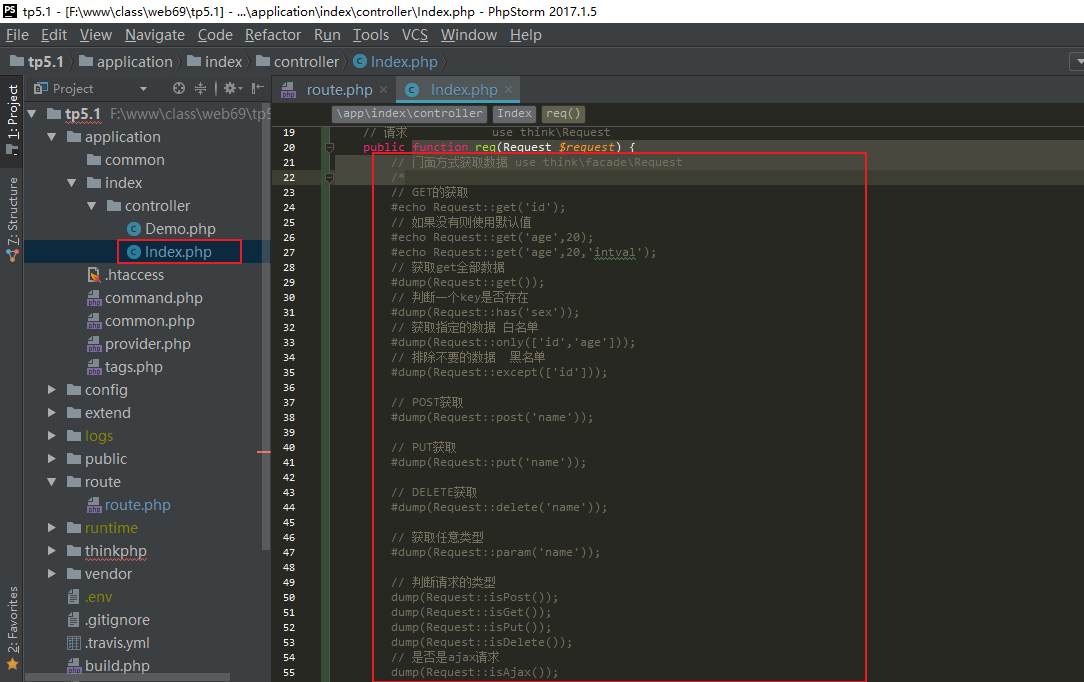


路由

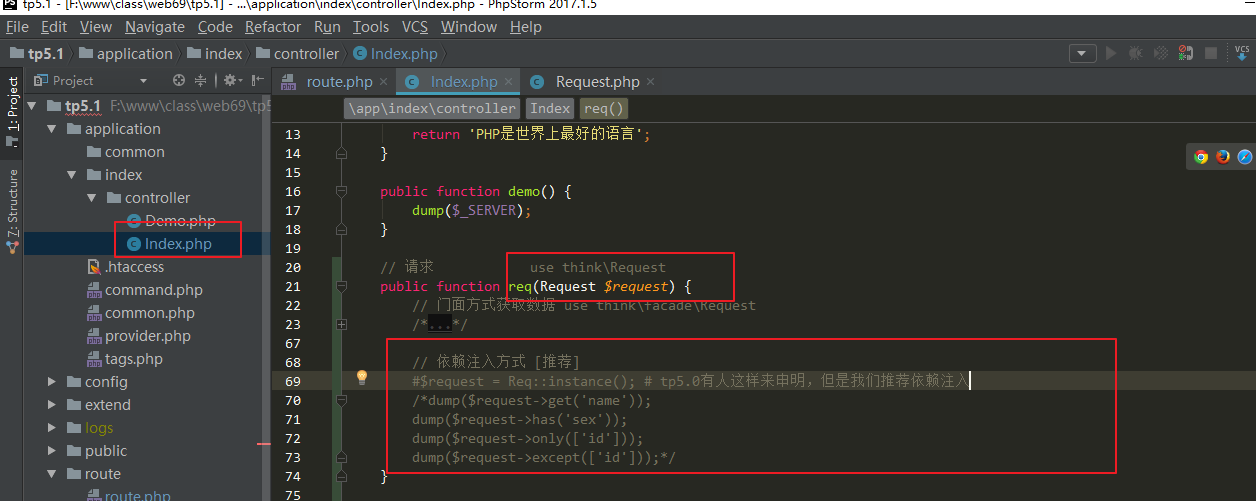


控制器

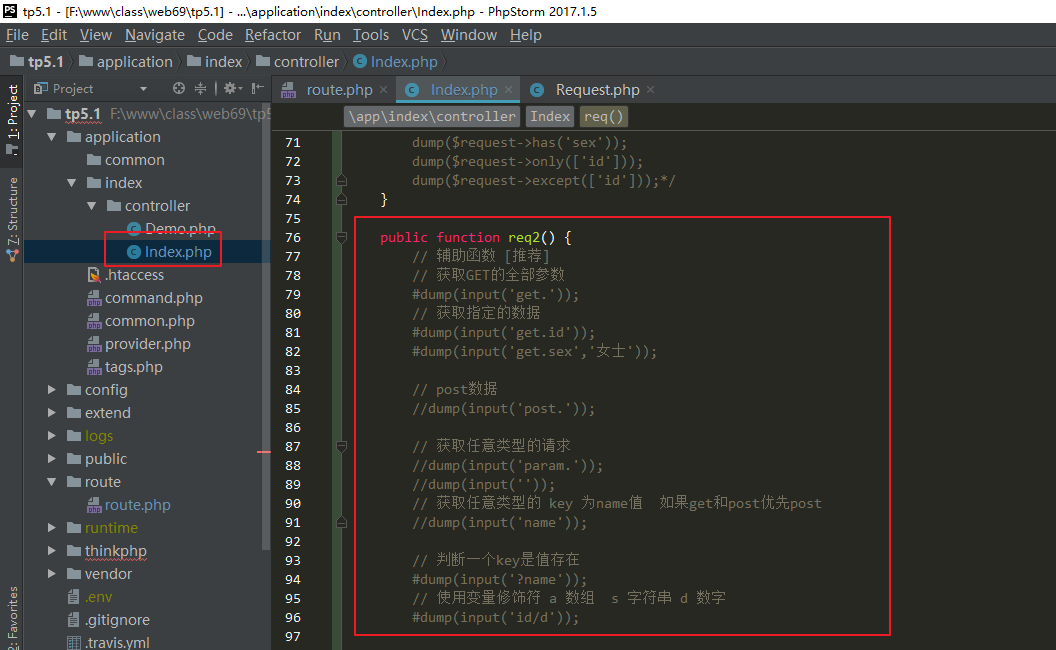
门面的方式



依赖注入方式获取【推荐】



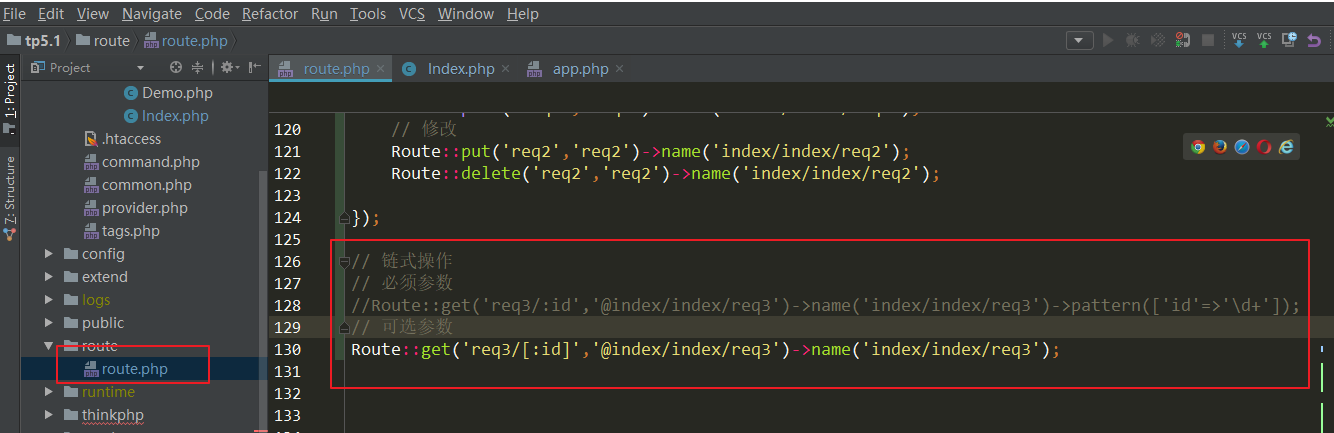
辅助函数【推荐】

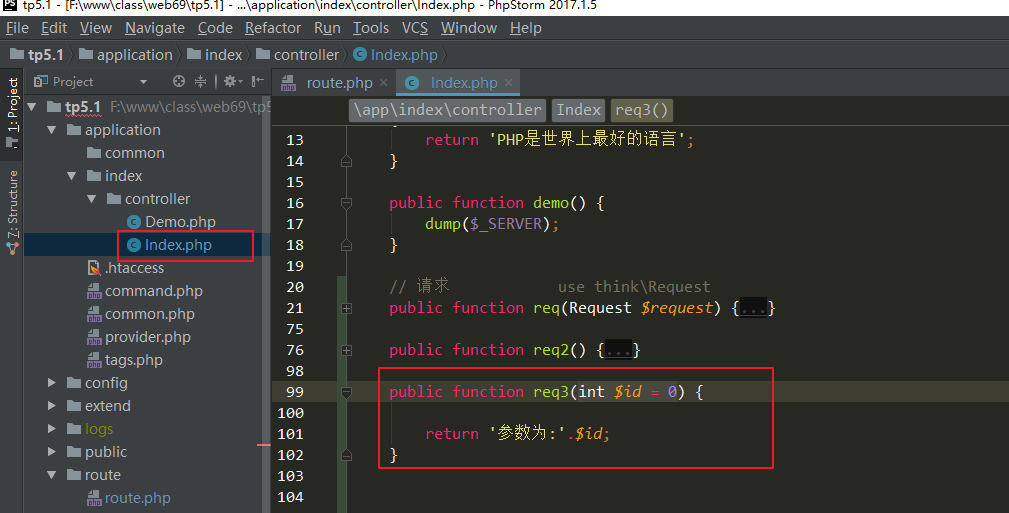


## 1.2、参数绑定

参数绑定是把当前请求的路由参数作为操作方法的参数直接传入，参数绑定并不区分请求类型。





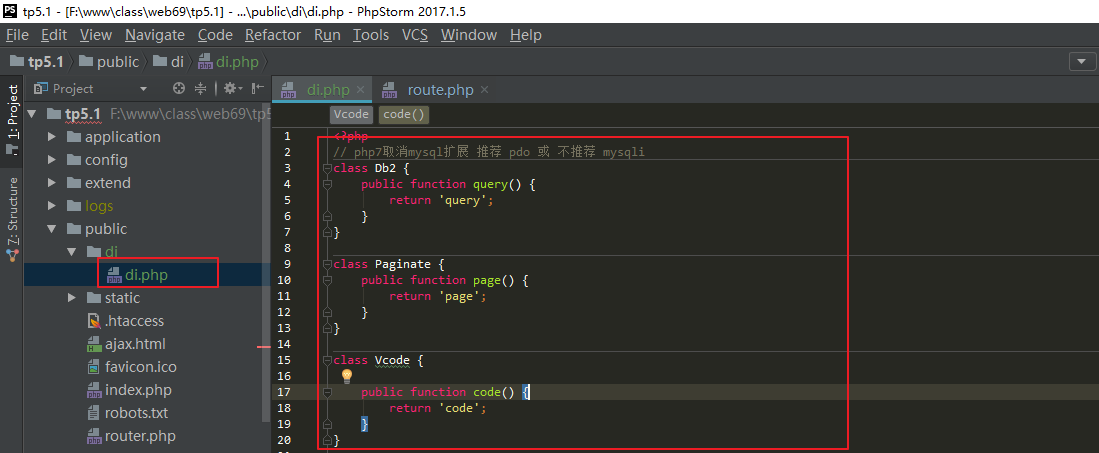


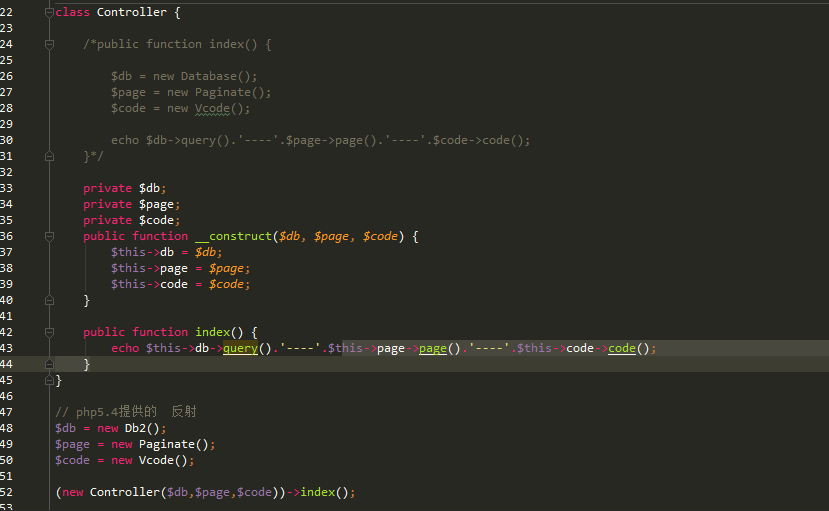
## 1.3、依赖注入

依赖注入是一种软件设计思想，在传统软件中，上层代码依赖于下层代码，当下层代码有所改动时，上层代码也要相应进行改动，因此维护成本较高。而依赖注入原则的思想是，上层不应该依赖下层，应依赖接口。意为上层代码定义接口，下层代码实现该接口，从而使得下层依赖于上层接口，降低耦合度，提高系统弹性。

控制反转【IOC】

依赖注入【DI】





# 二、响应

## 2.1、输出响应

return 'PHP是世界上最好的语言';

## 2.2、json数据返回

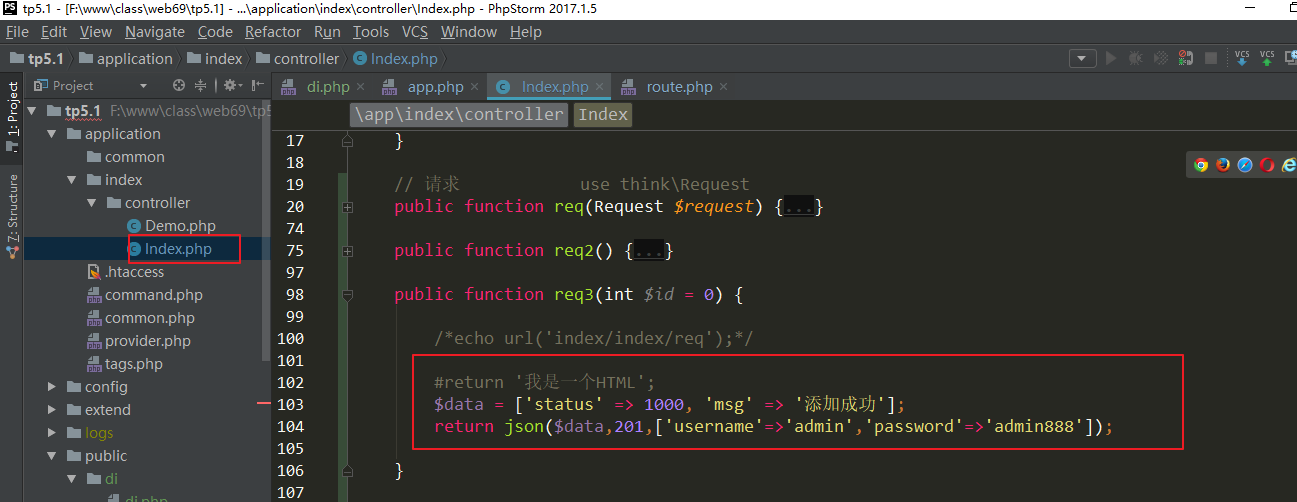
由于默认是输出Html输出，所以直接以html页面方式输出响应内容，但也可以设置默认输出为json格式 config/app.php

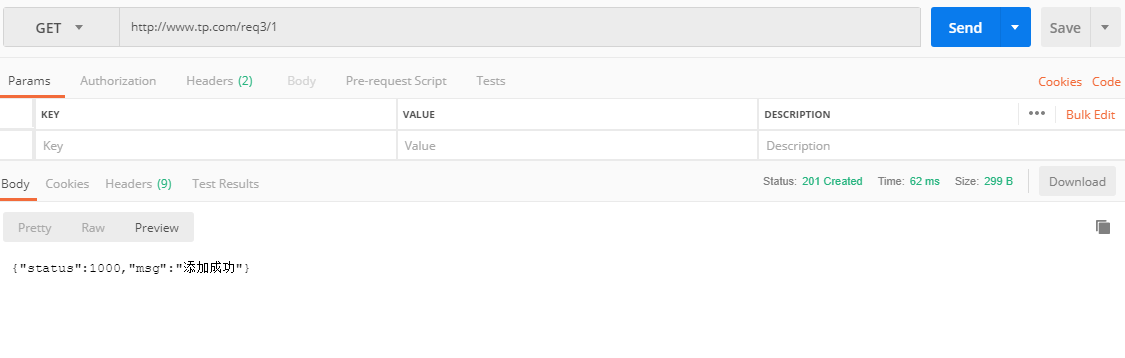
'default\_return\_type' => 'json',

返回json数据

return json($data,http状态码);

return json($data)->code(201)->header(['Content-Type' => 'application/json']);

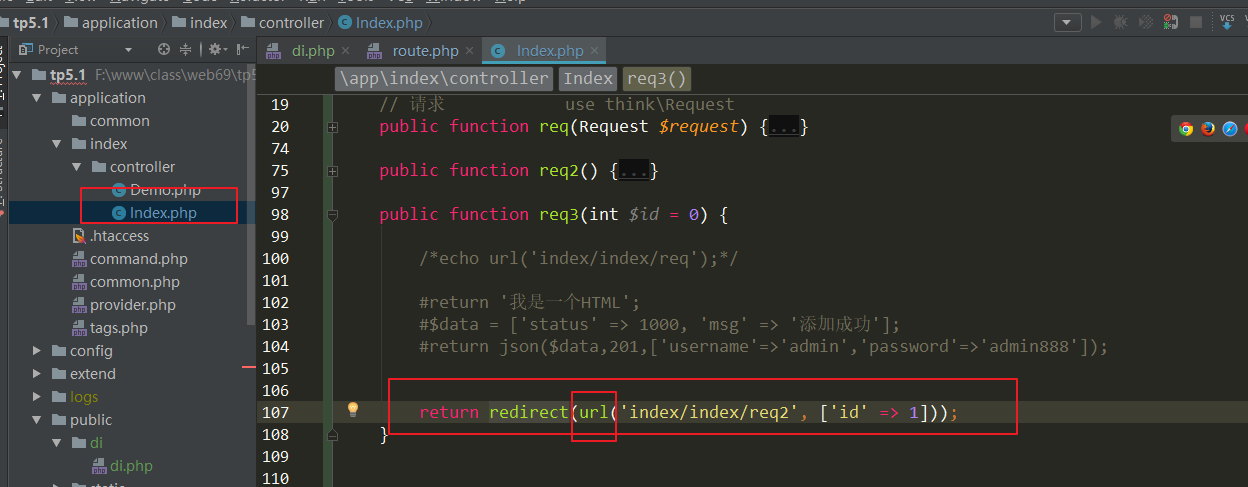




## 2.3、重定向

可以使用redirect助手函数进行重定向

redirect('地址或方法名',数组参数);



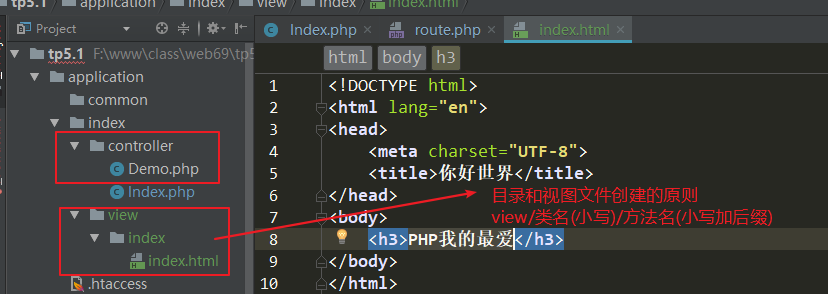
# 三、视图

## 3.1、视图的组成

此处所说的视图就是MVC中所说的V层，视图层也要展示层。thinkphp中视图层是由HTML模板文件组成的。

## 3.2、模板的定义

为了对模板文件更加有效的管理，ThinkPHP对模板文件进行目录划分，默认的模板文件定义规则是：视图目录(view)/控制器名（小写）/方法名（小写）+ 模板后缀(框架的默认视图文件后缀是.html)



## 3.3、模板渲染和赋值

控制器中指定渲染的模板文件

在控制器中使用

方法一：

return $this->fetch('[模板文件]'[,'模板变量（数组）']);

方法二：

return view('[模板文件]'[,'模板变量（数组）']); # 辅助函数

# 针对于自定义路由

return view('index@index/index');

模板赋值

方法一：了解

$this->assign('name','ThinkPHP');

方法二：推荐写法

以关联数组的方式在渲染模板方法第二个参数填写

return view('index@index/index',compact('aa','arr')); # 推荐写法

**全局赋值**

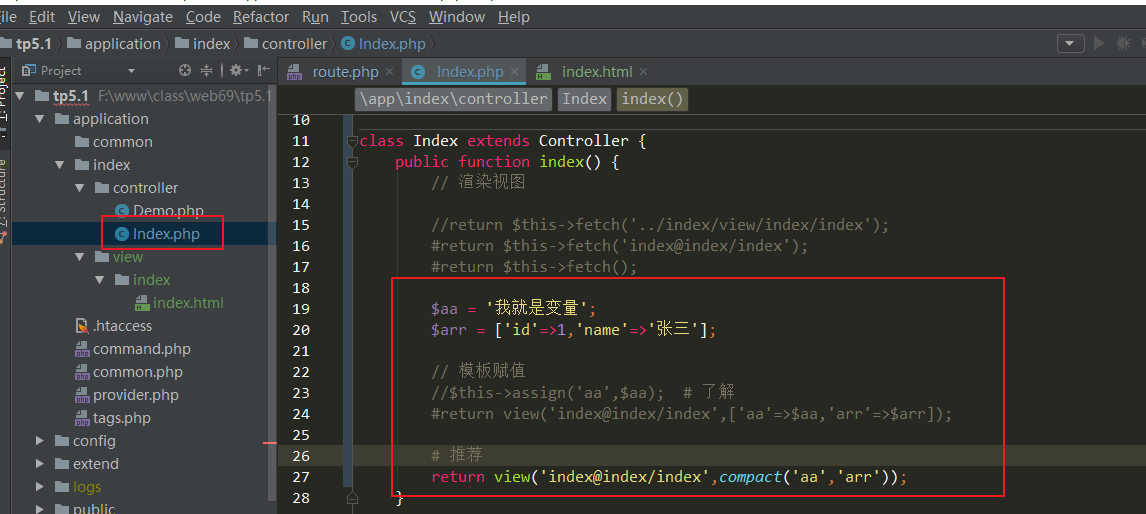
use think\facade\View;

# 赋值全局模板变量

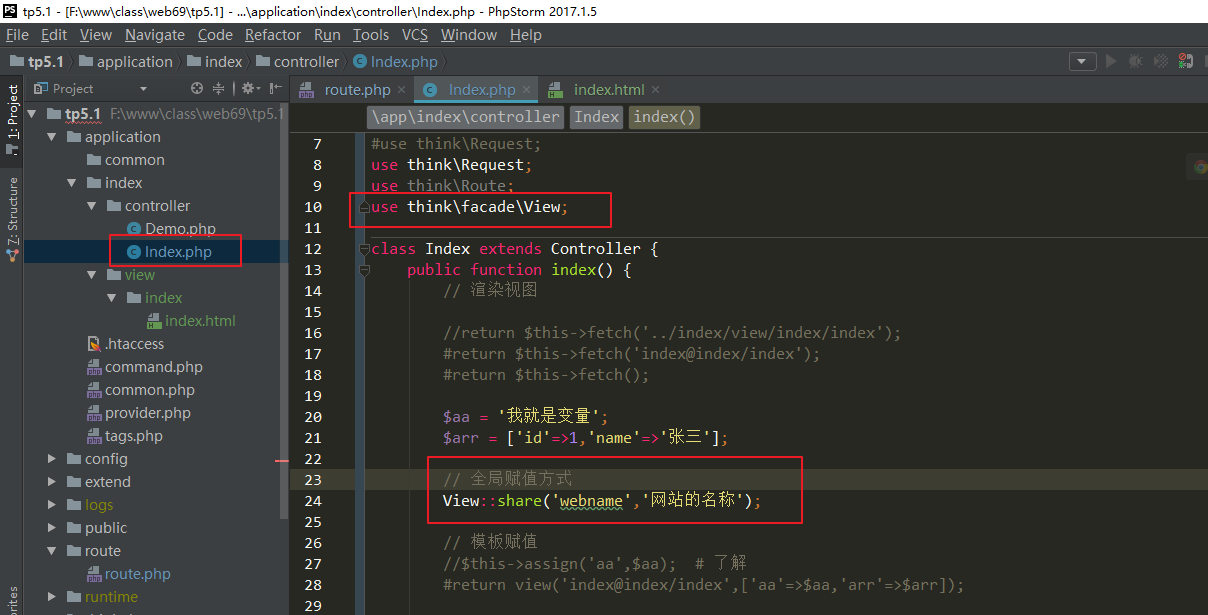
View::share('name','value');

# 或者批量赋值

View::share(['name1'=>'value','name2'=>'value2']);

s

全局赋值



总结：

渲染 view 函数

赋值 局部 compact() 全局 think\facade\View::share();

## 3.4、使用函数

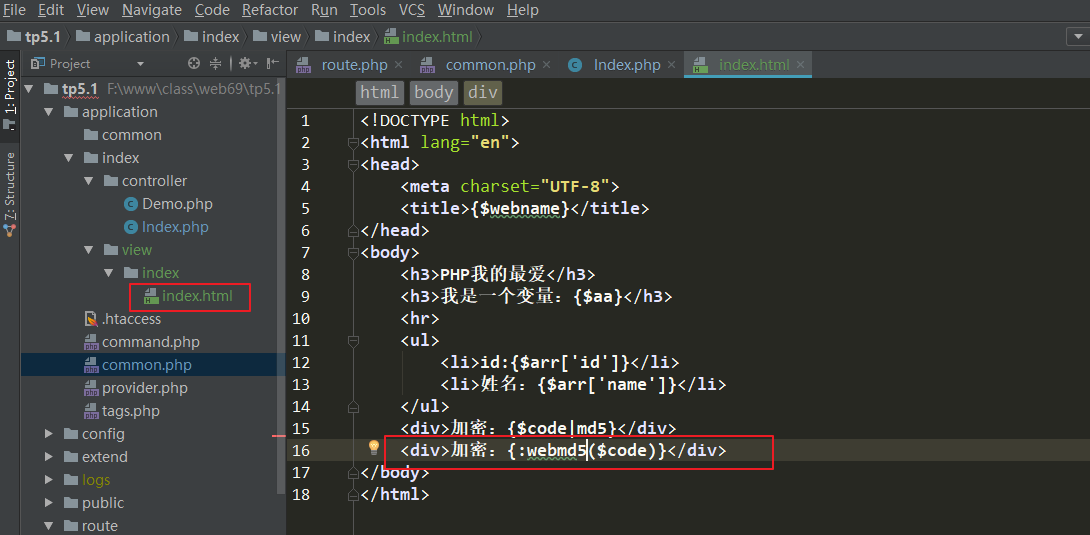
需要对模板输出使用函数进行过滤或其它处理的时候，就可以使用一下函数来解决。

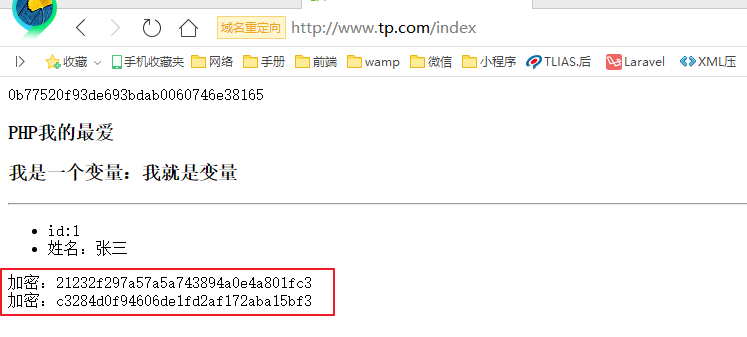
{$password|md5}

当然也可以写成如下的写法，推荐

{:md5($password)}

注：不但用php系统提供函数，还可以用tp提供的函数，还可以用户在tp规定的文件中定义的写的函数





## 3.5、原样输出

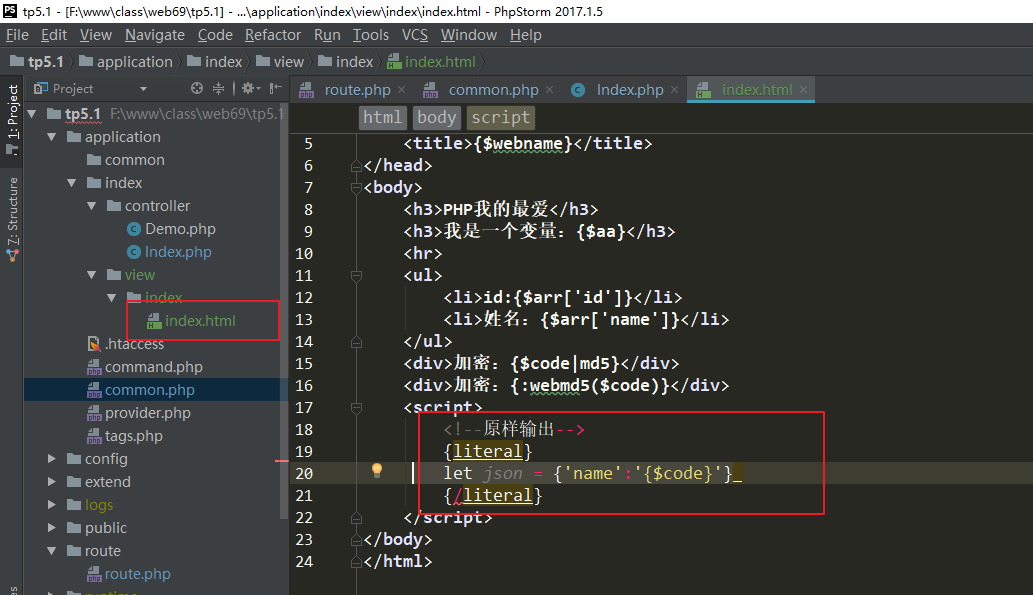
因thinkphp5的标签语法和js的一致，所以有的时候，可能会使用到它。

{literal}

Hello,{$name}！

{/literal}

上面的{$name}标签被literal标签包含，因此并不会被模板引擎解析，而是保持原样输出。





## 3.6、循环标签【重点】

# foreach 【推荐写法】

{foreach $list as $key=>$vo }

{$vo.id}:{$vo.name}

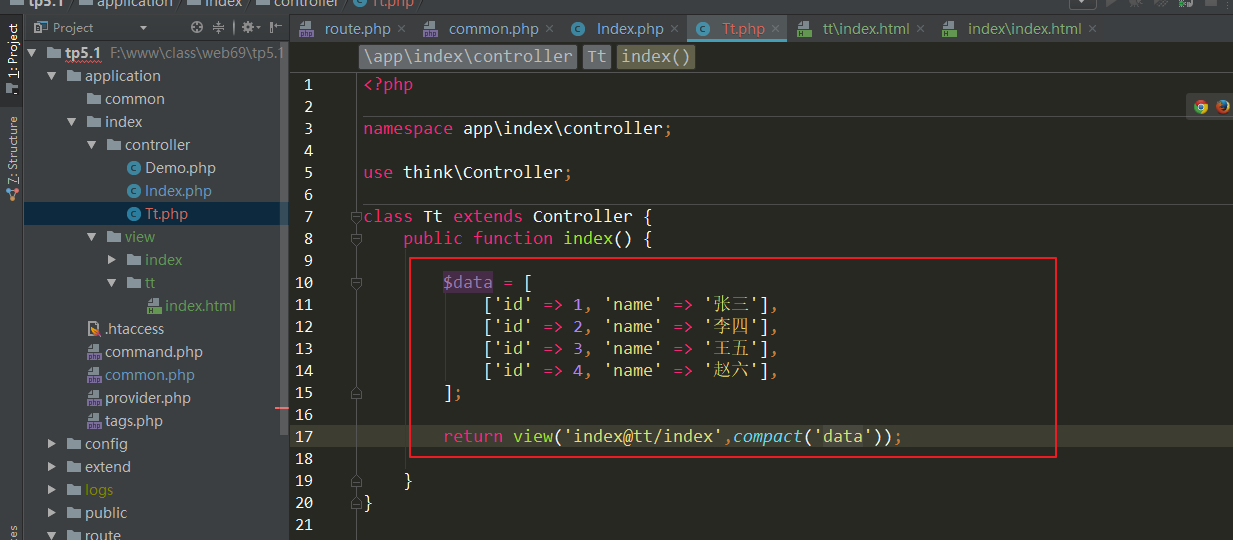
{/foreach}

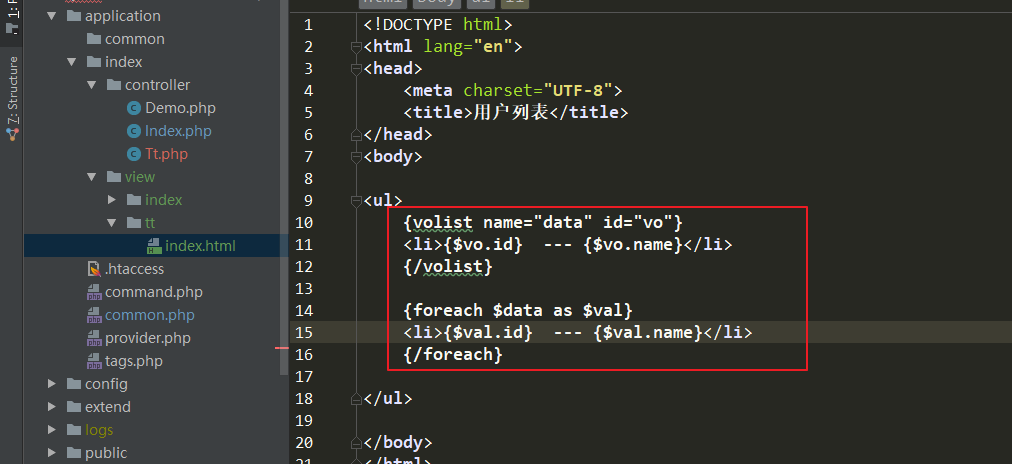
# volist

{volist name="list" id="vo"}

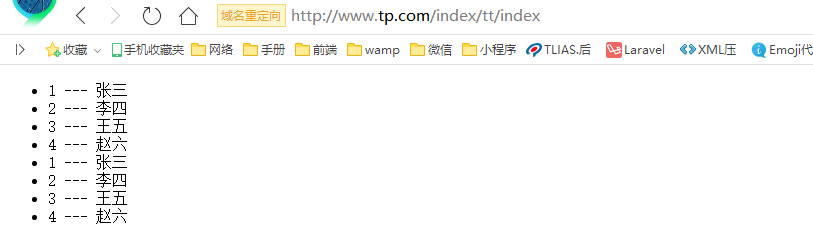
{$vo.id}:{$vo.name}<br/>

{/volist}





效果



## 3.7、条件判断

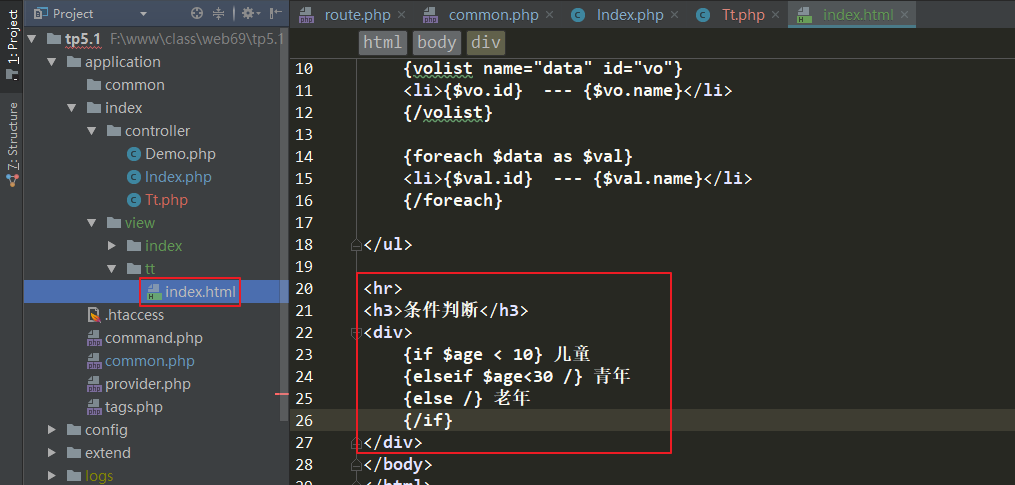
{if ( $name == 1) OR ( $name > 100) } value1

{elseif $name == 2 /} value2

{else /} value3

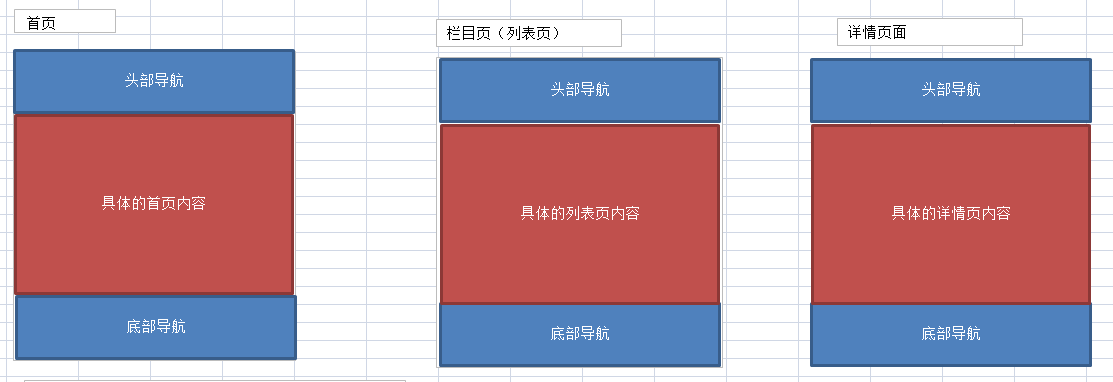
{/if}





## 3.8、模板继承

模板继承其实并不难理解，就好比类的继承一样，模板也可以定义一个基础模板（或者是布局），并且其中定义相关的区块（block），然后继承（extend）该基础模板的子模板中就可以对基础模板中定义的区块进行重载。

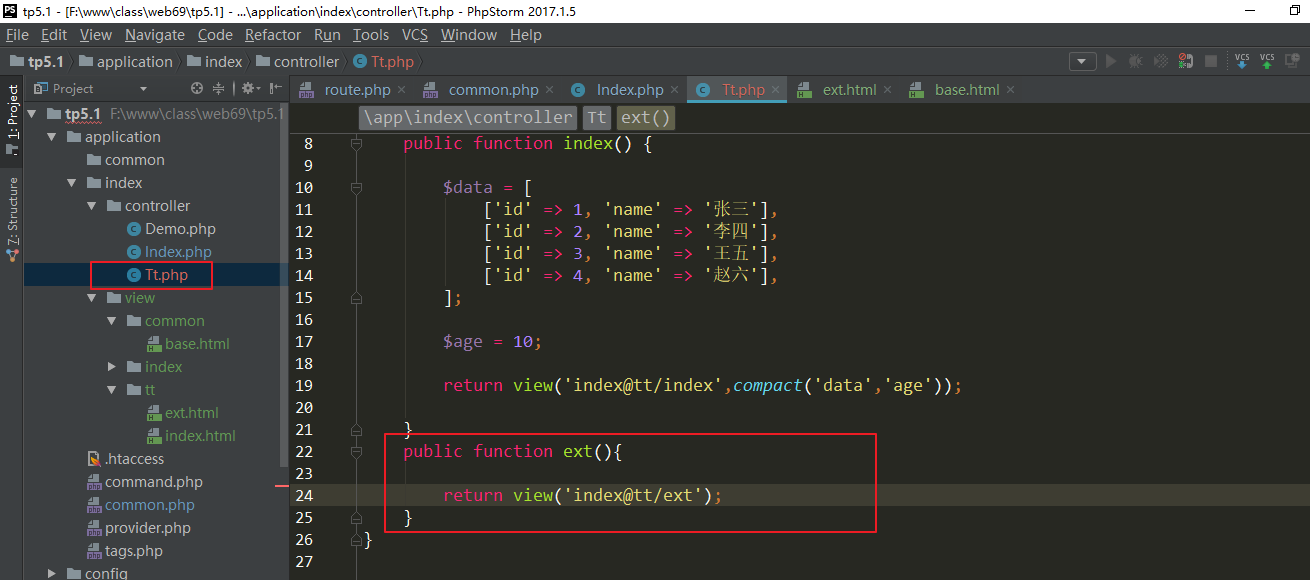


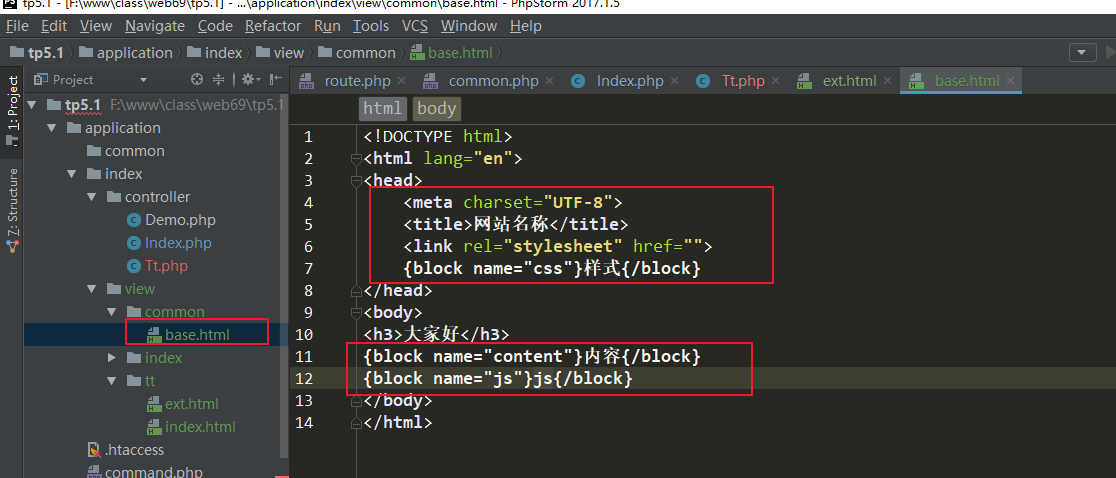
**定义一个基础模板**

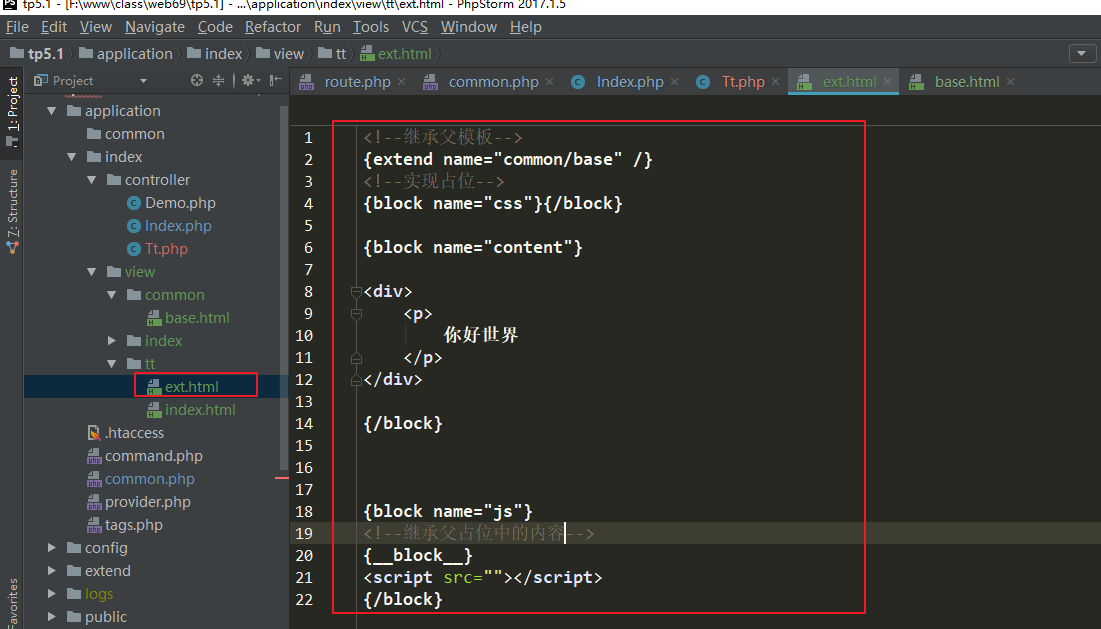


**在子模板（其实是当前操作的入口模板）中使用继承**









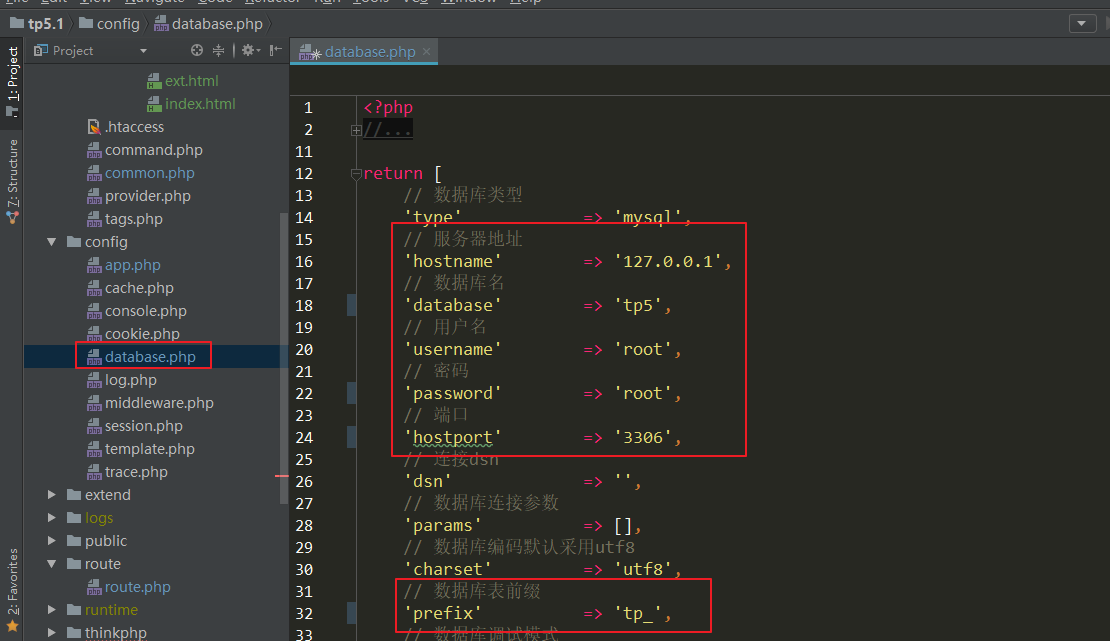


# 四、数据库操作

## 4.1、连接数据库

在应用配置目录或者模块配置目录下面的config/database.php中配置下面的数据库参数





注：配置好的，一定要检查mysql服务是否开启，同时也要检查pdo是否打开。

同时还是创建好对应的数据库和连接数据库的用户名和密码并确保他们可以连接上你们的mysql服务器。

## 4.2、执行原生sql语句

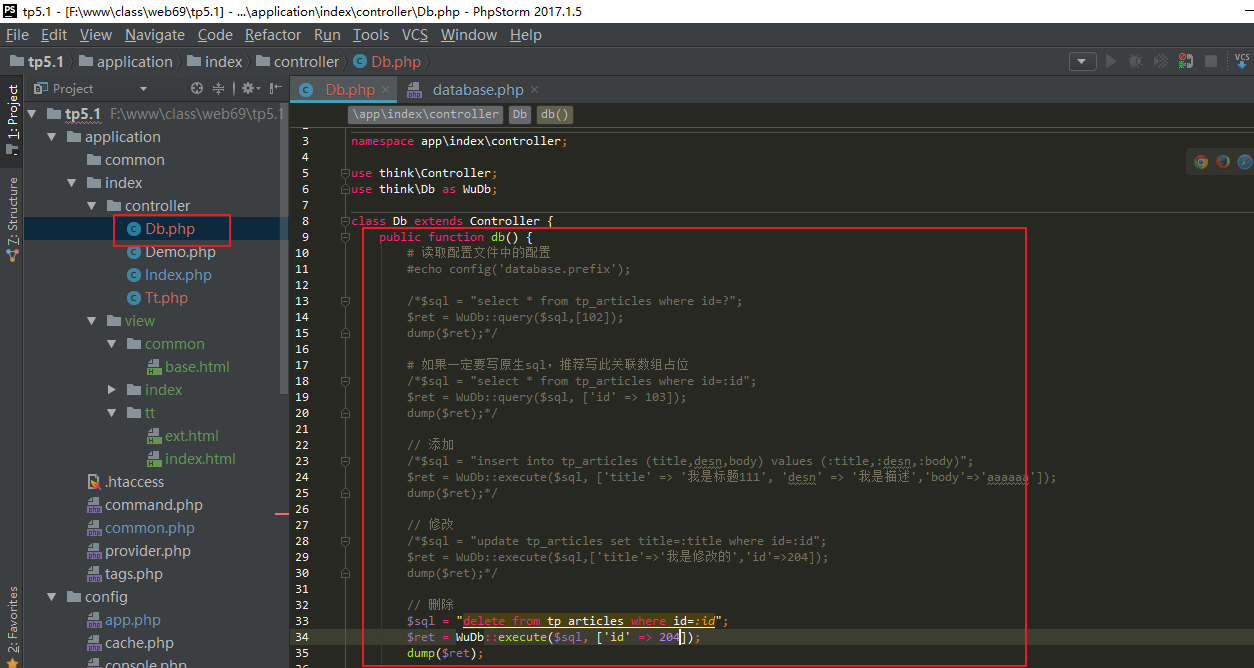
Db类支持原生SQL查询操作

# 查询

Db::query("select \* from think\_user where id=? AND status=?", [8, 1]);

# 添加、更新与删除

Db::execute("update think\_user set name=:name where status=:status", ['name' => 'thinkphp', 'status' => 1]);



## 4.3、添加数据Db

# 添加一条数据

$data = ['foo' => 'bar', 'bar' => 'foo'];

Db::name('user')->insert($data);

Db::name('user')->data($data)->insert();

db('user')->data($data)->insert();

# 添加数据后如果需要返回新增数据的自增主键

$userId = Db::name('user')->insertGetId($data);

# 添加多条数据

$data = [

['foo' => 'bar', 'bar' => 'foo'],

['foo' => 'bar1', 'bar' => 'foo1'],

['foo' => 'bar2', 'bar' => 'foo2']

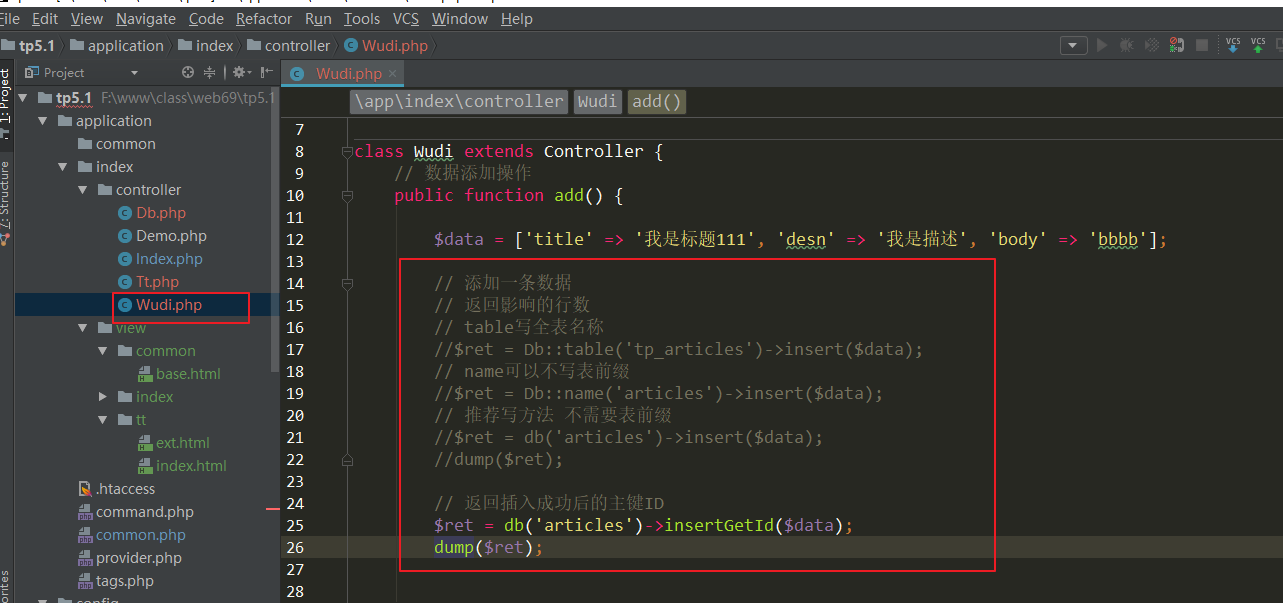
];

Db::name('user')->insertAll($data);

db('user')->insertAll($data);

Db::name('user')->data($data)->insertAll();

添加一条



添加多条



## 4.4、更新数据

更新数据一定要写条件

**db('user')->where('id', 1)->update(['name' => 'thinkphp']);**

db('user')->where('id', 1)->data(['name' => 'thinkphp'])->update();

Db::name('user')->update(['name' => 'thinkphp','id'=>1]);

# 5.1.7之后还支持了Db::raw写法

Db::name('user')

->where('id', 1)

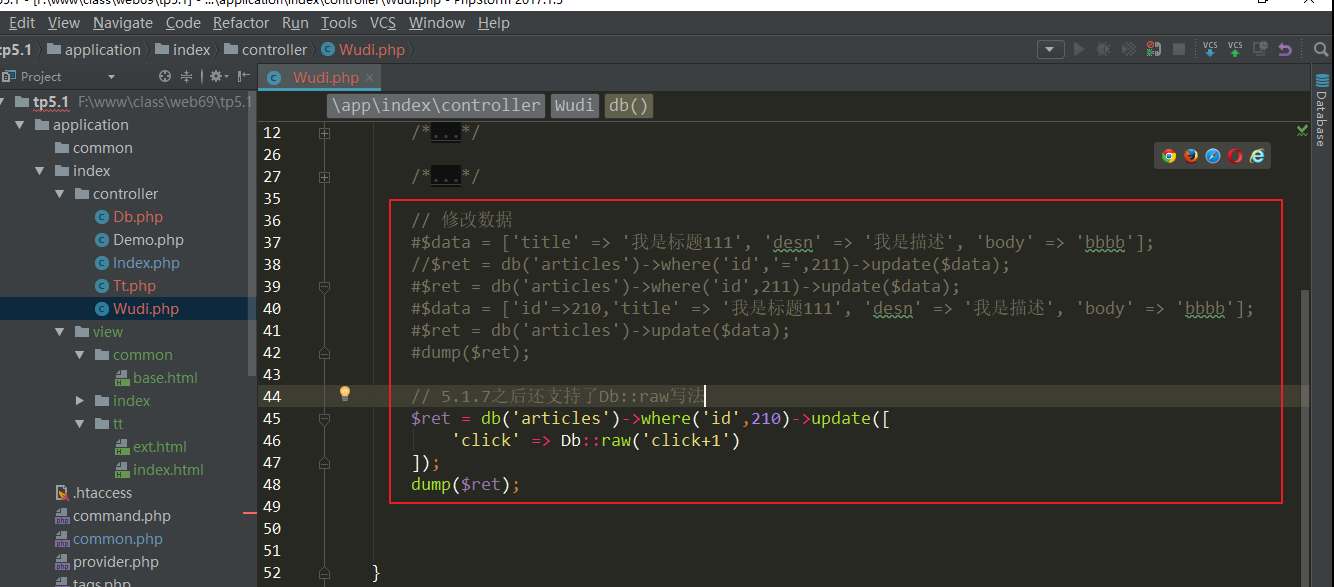
->update([

'name' => Db::raw('UPPER(name)'),

'score' => Db::raw('score-3'),

'read\_time' => Db::raw('read\_time+1')

]);



## 4.5、删除数据

# 根据主键删除

Db::table('think\_user')->delete(1);

Db::table('think\_user')->delete([1,2,3]);

# 条件删除

Db::table('think\_user')->where('id',1)->delete();

Db::table('think\_user')->where('id','<',10)->delete();

# 无条件删除所有数据 小心去用

Db::name('user')->delete(true);

# 软删除数据 使用delete\_time字段标记删除 逻辑删除

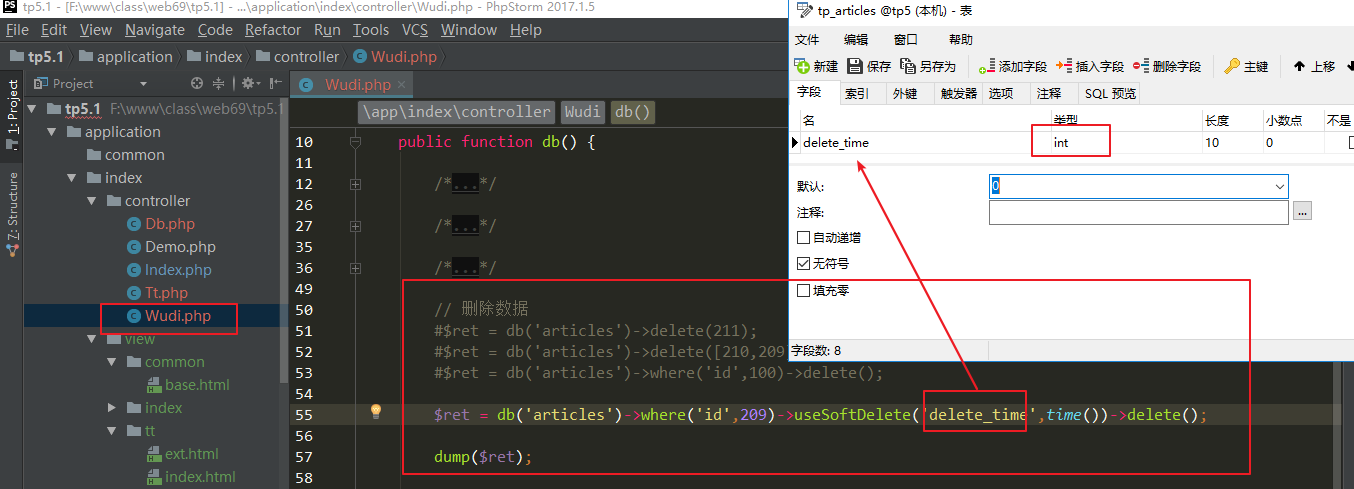
# 软删除，必须要表中要delete\_time字段

Db::name('user')

->where('id', 1)

->useSoftDelete('delete\_time',time())

->delete();



## 4.6、查询数据

# 查询一条数据

db('user')->find(主键ID);

Db::name('user')->where('id',1)->find();

# 如果希望在没有找到数据后抛出异常可以使用

Db::name('user')->where('id',1)->findOrFail();

# 查询多条记录

Db::name('user')->where('status',1)->select();

# 如果希望在没有找到数据后抛出异常可以使用

Db::name('user')->where('status',1)->selectOrFail();

# 查询某个字段的值可以用

Db::name('user')->where('id',1)->value('name');

# 查询某一列的值可以用

Db::name('user')->where('id',1)->column('name');

# 排序并获取指定记录条数

Db::name('user')->order('id', 'desc')->limit(0,10)->select();

# 聚合查询

Db::name('user')->count();

Db::name('user')->max('score');

Db::name('user')->where('score', '>', 0)->min('score');

Db::name('user')->avg('score');

Db::name('user')->where('id',10)->sum('score');

# 批量（字段）查询

$map = [

['name', 'like', 'thinkphp%'],

['title', 'like', '%thinkphp'],

['id', '>', 0],

];

Db::table('think\_user')

->where([ $map ])

->where('status',1)

->select();

SELECT \* FROM `think\_user` WHERE ( `name` LIKE 'thinkphp%' AND `title` LIKE '%thinkphp' AND `id` > 0 ) AND `status` = '1'

# 闭包查询

$name = 'thinkphp';

$id = 10;

Db::table('think\_user')->where(function ($query) use($name, $id) {

$query->where('name', $name)->whereOr('id', '>', $id);

})->select();

SELECT \* FROM `think\_user` WHERE ( `name` = 'thinkphp' OR `id` > 10 )

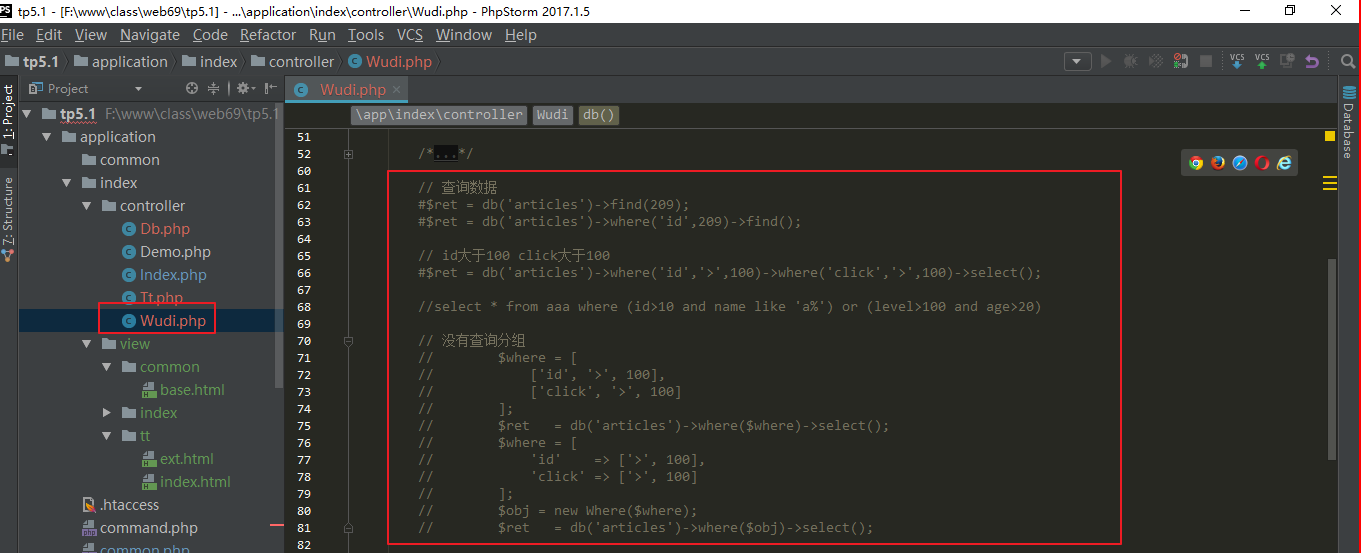
# 获取器 5.1.20之后才有

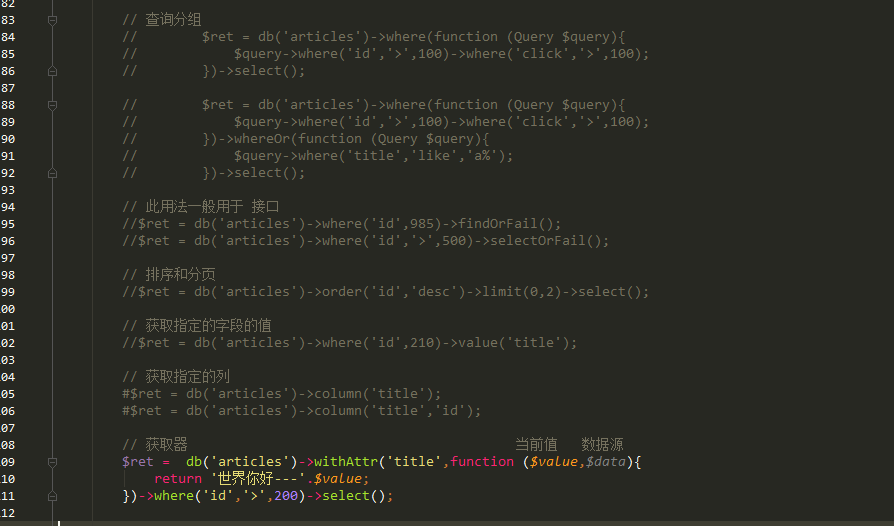
Db::name('user')->withAttr('name', function($value, $data) {

return strtolower($value);

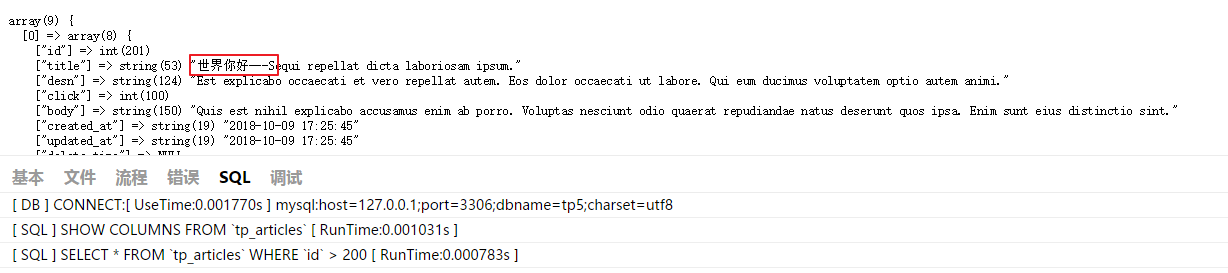
})->select();

上面的代码，查询的数据集数据中的name字段的值会统一进行小写转换





效果



# 五、模型操作

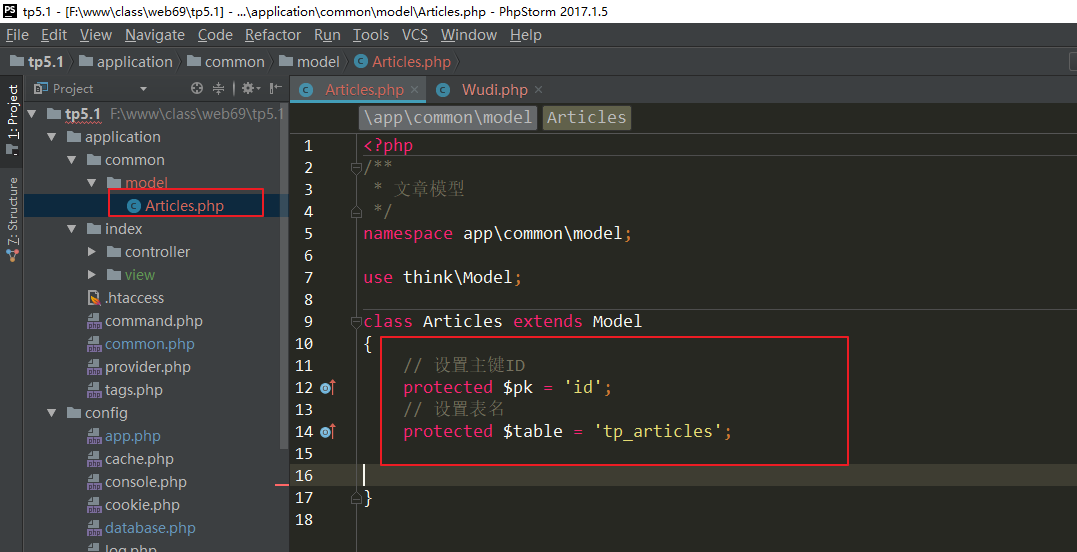
## 5.1、模型定义与模型设置

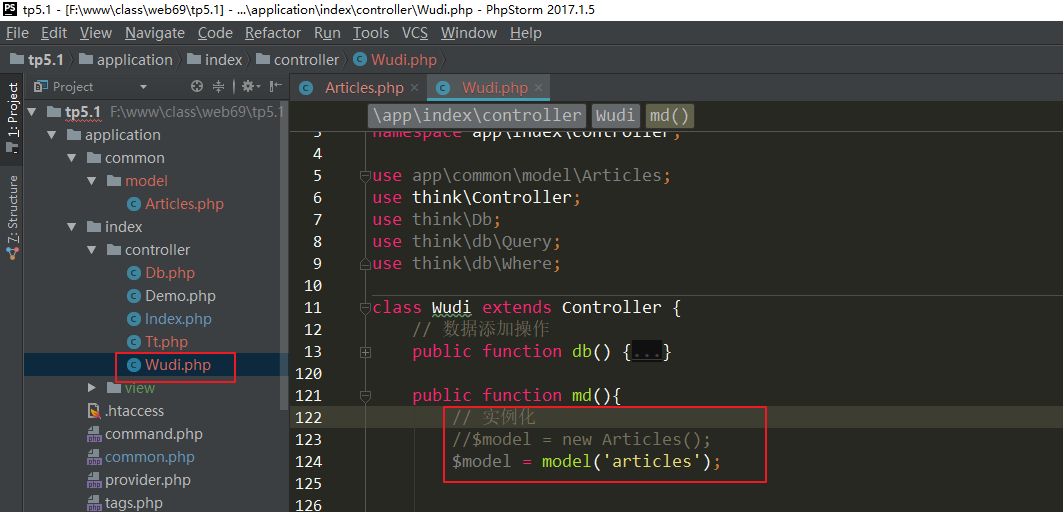
php think make:model 模块名/模型名(首字母大写)

# 非必须

protected $pk = 'uid'; # 设置主键名称

protected $table = 'think\_user'; #设置当前模型对应的完整数据表名称





## 5.2、添加数据

# 添加一条数据

# 方法1

$user = new User;

$user->name = 'thinkphp';

$user->email = 'thinkphp@qq.com';

$user->save();

# 方法2

$user = new User;

$user->save([

'name' => 'thinkphp',

'email' => 'thinkphp@qq.com'

]);

// 过滤post数组中的非数据表字段数据

$user->allowField(true)->save($\_POST);

$user->allowField([要插入的字段,要插入的字段])->save($\_POST);

# 方法3

$user = User::create([

'name' => 'thinkphp',

'email' => 'thinkphp@qq.com'

]);

echo $user->name;

# 添加多条记录

$user = new User;

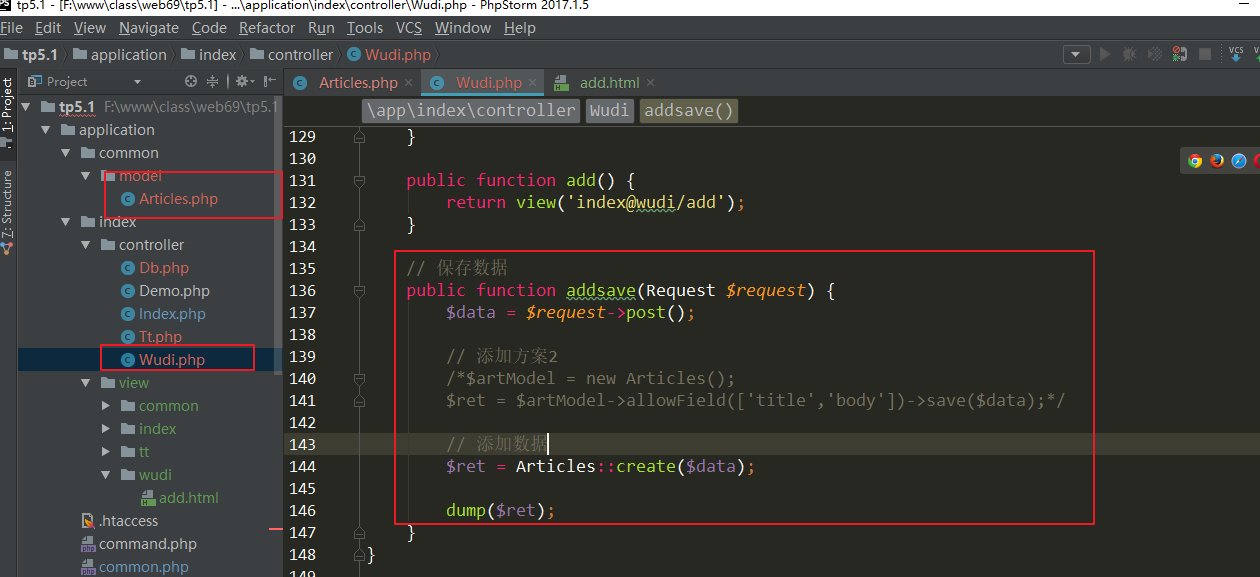
$list = [

['name'=>'thinkphp','email'=>'thinkphp@qq.com'],

['name'=>'onethink','email'=>'onethink@qq.com']

];

$user->saveAll($list);



## 5.3、更新数据

# 方法1

$user = User::get(1); find

$user->name = 'thinkphp';

$user->email = 'thinkphp@qq.com';

$user->save();

# 方法2

$user = new User;

// save方法第二个参数为更新条件

$user->save([

'name' => 'thinkphp',

'email' => 'thinkphp@qq.com'

],['id' => 1]);

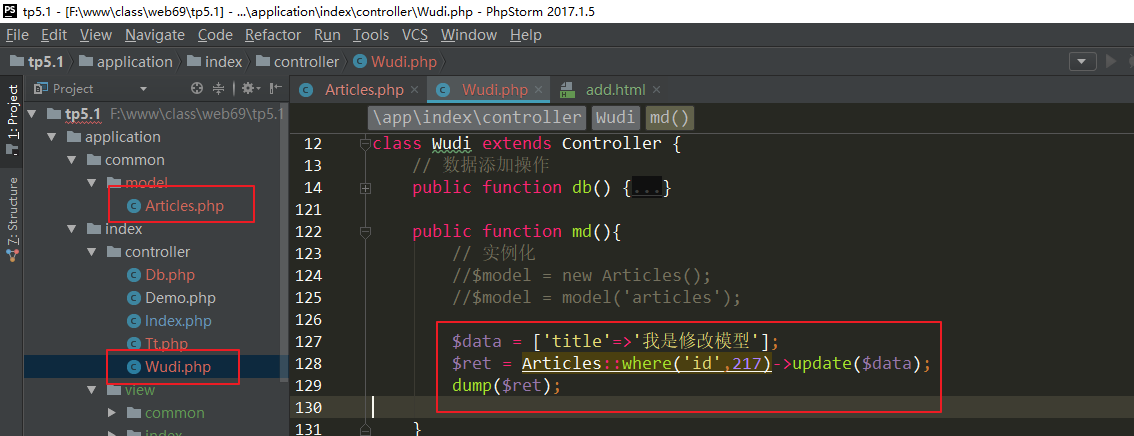
$user = new User;

// 过滤post数组中的非数据表字段数据

$user->allowField(true)->save($\_POST,['id' => 1]);

# 方法3

User::where('id', 1)->update(['name' => 'thinkphp']);



## 5.4、删除数据

# 方法1

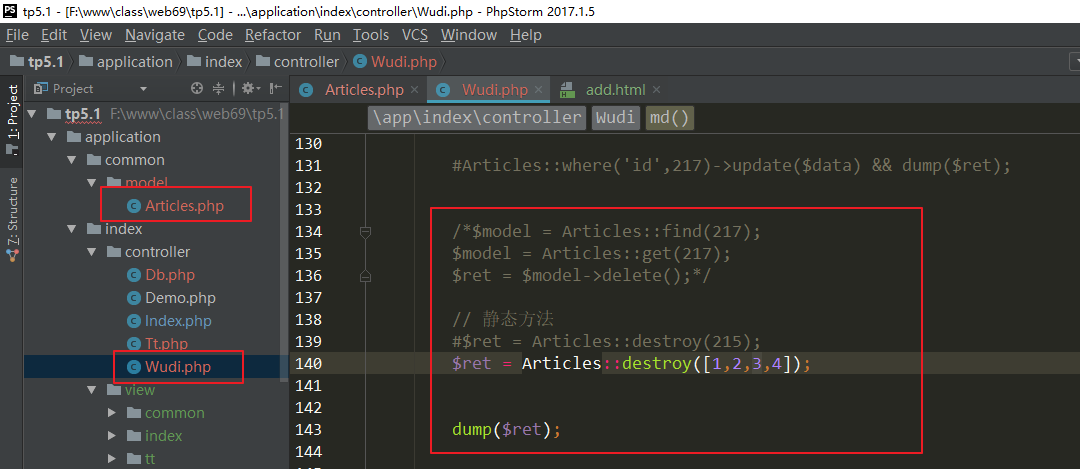
$user = User::get(1);

$user->delete();

# 方法2

User::destroy(1);

User::destroy([1,2,3]);

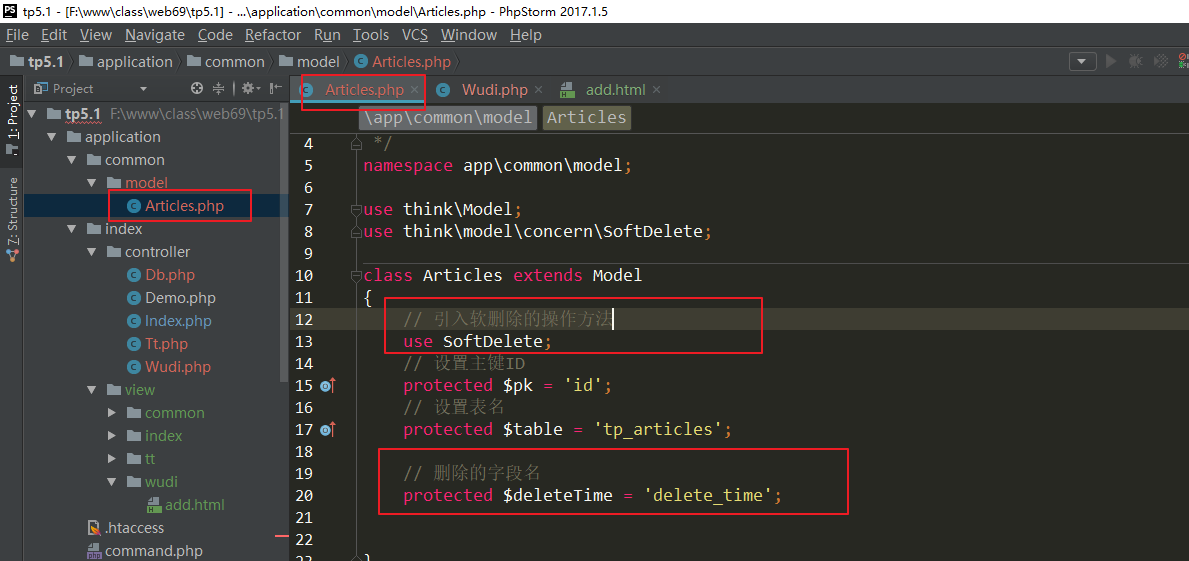


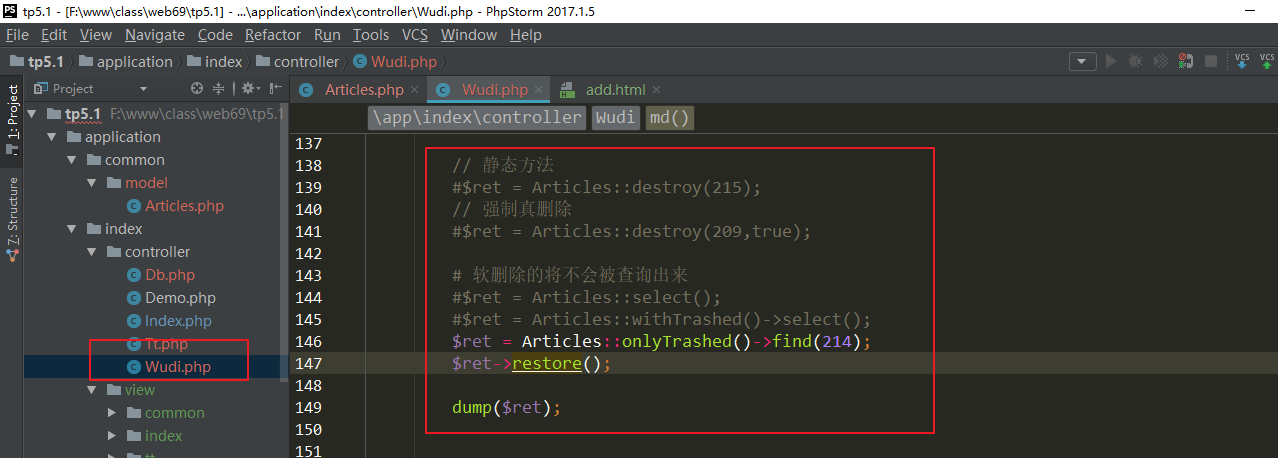
## 5.5、软删除

在实际项目中，对数据频繁使用删除操作会导致性能问题，软删除的作用就是把数据加上删除标记，而不是真正的删除，同时也便于需要的时候进行数据的恢复。









## 5.6、查询数据

# 查询单条记录

$user = User::where('name', 'thinkphp')->find();

# 查询多条记录

$list = User::where('status', 1)->limit(3)->order('id', 'asc')->select();

# 获取某个字段或者某个列的值

// 获取某个用户的积分

User::where('id',10)->value('score');

// 获取某个列的所有值

User::where('status',1)->column('name');

# 动态查询

// 根据name字段查询用户

$user = User::getByName('thinkphp');